TCP mechanisms for improving the user experience for time-dependent thin-stream applications

Andreas Petlund, Kristian Evensen, Carsten Griwodz and Pål Halvorsen
Simula Research Laboratory, Norway
Department of Informatics, University of Oslo, Norway
Email: {apetlund, kristrev, griff, paalh}@simula.no

***Abstract*—A wide range of Internet-based services that use reliable transport protocols display what we call thin-stream properties. This means that the application sends data with such a low rate that the retransmission mechanisms of the transport protocol are not fully effective [7]. In time-dependent scenarios where the user experience depends on the data delivery latency, packet loss can be devastating for the service quality. In order to reduce application-layer latency when packets are lost, we have implemented modifications to the TCP retransmission mechanisms in the Linux kernel. The changes are only active when thin-stream properties are detected, thus not affecting TCP behaviour when the stream is not thin. In this paper, we show the latency improvements from these thin-stream modifications. We have tested several thin-stream applications like remote terminals (SSH) and audio conferencing (Skype), and we evaluate the user experience with and without the TCP modifications. Our experimental results show that our modifications allow TCP to recover earlier from packet loss. Furthermore, user surveys indicate that the majority of users easily detect improvements in the perceived quality of the tested applications.**

***Index Terms*—TCP, thin streams, latency reduction, user evaluation**

I. Introduction

Reliable transport protocols that are in widespread use today, like TCP, are primarily designed to support connections with high throughput efficiently. The aim is to move lots of data from one place to another as fast as possible (like HTTP, FTP etc.). However, a wide range of networked applications do not follow this pattern. Important examples include interactive applications where small amounts of data are transmitted sporadically, e.g., multiplayer online games, audio conferences, and remote terminals. Due to the highly interactive nature of these applications, they depend on timely delivery of data. For example, data delivery latency in audio conferences should stay below 150-200 ms to achieve user satisfaction [8], and online games require latencies in the range of 100-1000 ms, depending on the type of game, to be able to provide a satisfying experience to the players [5]. Furthermore, when dealing with stock exchange systems, even low delays in delivery may constitute a large potential economic disadvantage.

To support different requirements for timeliness, distributed interactive applications have historically been developed for use either with transport protocols that can provide per-stream quality of service (QoS) guarantees, or with protocols that at least allow the sender to determine the timing of the data transmission. The QoS protocols for the first approach have not become widely available. The use of UDP (the protocol that provides for the second approach) has been widely criticized for its lack of congestion control mechanisms. Consequently, many distributed interactive applications today are built to work with TCP, and many applications that use UDP (in spite of said criticism) fall back to using TCP if, for example, a firewall is blocking UDP traffic. The disadvantage of using TCP is that applications that generate what we call "thin streams", can experience severe delays when TCP recovers from loss. A stream is called "thin" if at least one of the following criteria is fulfilled: **a)** The packet interarrival time (IAT) is too high to be able to trigger fast retransmissions. **b)** The size of most packets are far below the maximum segment size (MSS). The occasional high delays for thin streams are caused by the retransmission mechanisms common for reliable transport protocols [7].

In this paper, we discuss the performance of a set of time-dependent thin-stream applications by referring to analysis of packet traces from sessions using the respective applications. We have analyzed the traces with respect to inter-arrival time between packets, packet size and consequently bandwidth requirement. The transmission characteristics are then discussed with respect to the shortcomings identified in standard TCP congestion control mechanisms when dealing with thin streams. Moreover, we have implemented a set of modifications to TCP with the goal of reducing the data delivery latency for thin stream applications when retransmissions are necessary. Two representative examples (Skype using TCP fallback and SSH sessions) are tested with and without our modifications. First, we analyze the packet (data) arrival latencies at both the transport- and application layer. In addition, subjective evaluations are performed by a group of people in order to provide an indication of whether it is possible to perceive a difference in performance for the applications in question. The results from the user surveys are then compared to the statistical results gleaned from packet traces of the same applications. In summary, our results show that the proposed modifications allow TCP to recover from packet loss more rapidly, and the user surveys indicate that the majority of users easily detect improvements according to the earlier data recovery.

application (platform)	prot-ocol	payload size (bytes)			packet interarrival time (ms)						avg. bandwidth requirement	
		average	min	max	average	median	min	max	percentiles 1%	percentiles 99%	(pps)	(bps)
Anarchy Online	TCP	98	8	1333	632	449	7	17032	83	4195	1.582	2168
BZFlag	TCP	30	4	1448	24	0	0	540	0	151	41.667	31370
Casa	TCP	175	93	572	7287	307	305	29898	305	29898	0.137	269
Windows remote desktop	TCP	111	8	1417	318	159	1	12254	2	3892	3.145	4497
Skype (2 users)	UDP	111	11	316	30	24	0	20015	18	44	33.333	37906
Skype (2 users)	TCP	236	14	1267	34	40	0	1671	4	80	29.412	69296
SSH text session	TCP	48	16	752	323	159	0	76610	32	3616	3.096	2825

TABLE I
EXAMPLES OF THIN STREAM PACKET STATISTICS BASED ON ANALYSIS OF PACKET TRACES.

II. INTERACTIVE THIN STREAMS

A large selection of networked interactive applications displays traffic patterns that we call thin streams. These applications generate data in such a way that packets are small and/or have high packet interarrival time. Popular examples include multiplayer online games, audio conferences, sensor networks, remote terminals, control systems, virtual reality systems, augmented reality systems and stock exchange systems. In order to understand thin-stream traffic better, we analyzed packet traces from example applications representing several scenarios (see some examples in table I). Our results show that thin-stream applications may have difficulties providing proper service when they are sent using established reliable transport protocols (like TCP [7] or SCTP [10]). Below, we take a closer look at two widely used examples before discussing the TCP shortcomings in this scenario.

A. Audio Conferencing

Audio conferencing with real-time delivery of voice data across the network is an example of a class of applications that uses thin data streams and has a strict timeliness requirement due to its interactive nature. Nowadays, audio chat is typically included in virtual environments, and IP telephony is increasingly common. For coding and compression, many VoIP telephone systems use the G.7xx audio compression formats recommended by ITU-T where, for example, G.711 and G.729 have a bandwidth requirement of 64 and 8 Kbps, respectively. The packet size is determined by the packet transmission cycle (typically in the area of a few tens of ms, resulting in packet sizes of around 80 to 320 bytes for G.711).

Skype [2] is a well-known conferencing service, with several million registered users, that communicates on the (best effort) Internet. We have analyzed several Skype sessions, and we see that this application shares the characteristics of IP telephony. The packets are small and the bandwidth low. Table I shows two examples of analyzed dumps from Skype conferencing traffic. Using UDP, the packet payload size for our trace is very low with a maximum payload of 316 bytes. The IAT between packets averages to 30 ms, which qualifies it as a thin-stream candidate. The second Skype example in table I is from a session where UDP has been firewalled, and TCP is used as a fallback option. We can see that there is a greater variation in payload size (probably due to TCP bundling upon retransmission), but still a low average payload compared to the maximum segment size. The average interarrival time is approximately the same as for the UDP dump. The result is that Skype over TCP seems to require a higher bandwidth than Skype over UDP. The bandwidth requirement is still low for both transport protocols, though.

To enable satisfactory interaction in audio conferencing applications, ITU-T defines guidelines for the one-way transmission time. These guidelines indicate that users begin to get dissatisfied when the delay exceeds 150-200 ms and that the maximum delay should not exceed 400 ms [8].

B. Remote Terminals

Windows Remote Desktop (using the remote desktop protocol (RDP) is an application used by thin client solutions or for remote control of computers. Analysis of packet traces from an RDP session indicates that this traffic also qualifies as a thin stream. If second-long delays occur due to retransmissions, this will result in visual delay for the user while performing actions on the remote computer.

Another way of working on a remote computer is the common protocol of secure shell (SSH). This is used to create an encrypted connection to a remote computer and control it, either using text console, or by forwarding graphical content. The analysed dump presented in table I is from a session where a text document is edited on the remote computer. We can observe that this stream also displays the thin-stream properties. The interarrival times are very similar to the RDP session, while the packet sizes are somewhat lower.

C. TCP Shortcomings

Griwodz et al [7] describe how, for streams of certain characteristics, TCP retransmission schemes result in higher latency when loss occurs. The reason is that the fast retransmit mechanism in TCP, which enables retransmission of lost segments before a timeout occurs, depends on feedback from the receiver (ACKs). The mechanism requires three duplicate acknowledgments (dupACKs) in order to initiate a fast retransmission [12], [4]. The reason for waiting until the third indication of loss is to avoid spurious retransmissions when reordering happens on the network. For thin stream scenarios, where interarrival times between sent packets are relatively high, the consequence is that many (or all) all retransmissions will be caused by timeouts. This is because there are seldom

enough packets in flight (packets on the wire) to generate the necessary feedback to trigger a fast retransmit. In addition, the retransmission timeout (RTO) is exponentially increased when multiple losses are detected (exponential backoff), which results in a still larger latency penalty. This mechanism is designed to ensure that an acceptable send rate is found, and to prevent a stream from exceeding it's fair share of the bandwidth resources. Thin streams, however, do not use their fair share, most of the time they stay within the worst-case throughput of 2 packets per RTT. The large IATs make it impossible for the thin stream to back off when packets are lost multiple times, resulting in a situation where the RTO value is very high, without any actual benefit with regard to resource distribution on the network. The result for the thin stream is that the retransmission is delayed by seconds (or even minutes) if segments is lost several times.

Applications that provide no interactive service may do well under such conditions. Thin-stream applications, however, are often interactive and may suffer from the extra delay. With these characteristics and strict latency requirements in mind, supporting thin-stream interactive applications is challenging. The task is made even more difficult by the fact that a significant characteristic of this type of application is its lack of resilience to network transmission delays. The data streams in the described scenarios are poorly supported by the existing TCP variations in Linux. Their shortcomings are that 1) they rarely trigger fast retransmissions, thus making timeouts the main cause of retransmissions, and 2) TCP-style congestion control does not apply because the stream cannot back off. Thus, since improvements for TCP have mainly focused on traditional thick stream applications like web and FTP download, new mechanisms are needed for the interactive time-dependent thin-stream scenario.

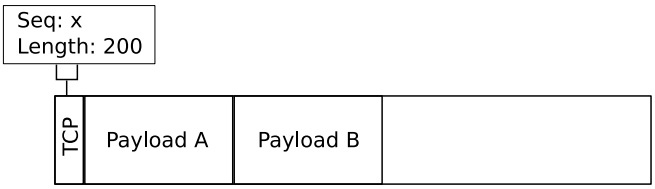
III. TCP Enhancements

The results of our earlier investigations of reliable transport protocols [7], [10], [6] show that it is important to distinguish between thick and thin streams with respect to latency. Where high-rate streams are only concerned with throughput, the perceived quality of most thin streams depends on timely delivery of data. Working from the assumption that there is potential for large performance gains by introducing modifications that tune TCP for thin streams, we have tested the combination of several such mechanisms on typical thin-stream applications. In short, if the kernel detects a thin stream, we trade a small amount of bandwidth for latency reduction by enabling modifications to TCP. The modifications are designed in such a way that they are transparent to the receiver (i.e., a server can run the modified TCP, and unmodified clients may still receive the benefits). Any client should therefore be able to receive the stream sent by the modified TCP version, regardless of operating system and version. The modifications are also transparent to the application running on top of TCP (they can be enabled by using a */proc* variable, and will thus be active for all networked applications). The following modifications have been implemented in the Linux kernel (v.2.6.23.8):

- **Removal of exponential backoff:** Since most thin streams send out packets with a high IAT, more or less all retransmissions are caused by timeouts. A timeout retransmission invokes exponential backoff, which doubles the time to wait for the next retransmission.
  If the number of packets in flight (i.e., unacknowledged packets) is less than the number required to trigger a fast retransmission, we remove the exponential factor. If more than four packets are on the wire, the possibility for triggering a fast retransmit increases, and therefore exponential backoff is employed as usual. Since the streams that gain from this modification are very thin, the increase in bandwidth consumption due to the removal of exponential backoff in these cases is very small. That is, the stream does still not have to use its allowed send window.
- **Faster Fast Retransmit:** Instead of having to wait several hundred milliseconds for a timeout retransmission and then suffer from the exponential backoff, it is much more desirable to trigger a fast retransmission. This requires the connection to wait for three duplicate acknowledgments (four acknowledgments of the same packets), which is not ideal for many thin stream scenarios. Due to the high IAT in our scenario, sending three packets often takes longer than the timeout.
  We have therefore reduced the number of required duplicate acknowledgments to one, provided that the number of packets in flight is less than four. Otherwise, the chance of receiving three dupACKs increases, and regular (three dupACKs) fast retransmit is employed.
- **Redundant Data Bundling:** As shown in table I, many thin-stream applications send small packets. As long as the combined packet size is less than the maximum segment size (MSS), we copy (bundle) data from unacknowledged packets in the send buffer into the new packet. If a retransmission occurs, as many of the remaining unacknowledged packets as possible are bundled with the retransmission. This increases the probability that a lost payload will be delivered already with the next packet. Figure 1 shows an example of how a previously transmitted data segment is bundled with the next packet. Notice how the sequence number stays the same while the packet length is increased. If packet a) is lost, the ACK from packet b) will ACK both segments, making a retransmission unnecessary.

As mentioned in the description of the first two modifications, they are only applied when the stream is thin. This is accomplished by defining a threshold for the number of packets in flight. Thus, to avoid that streams already using their fair share of the available resources (in accordance with TCP) consume even more using our proposed mechanisms, the proposed enhancements are only applied when thin stream properties are detected. Redundant data bundling (RDB), on the other hand, is limited by the IAT and packet size of the stream. If the packets are large, which is typical for bulk

(a) First sent packet.



(b) Second packet: Bundled data.

Fig. 1. Method of bundling unacknowledged data.

data transfer, bundles can not be made, and RDB will not do anything.

IV. Related work

A huge amount of work has been performed in the area of TCP performance. The vast majority of this work, however, addresses bulk transfer, whereas time-dependent thin-stream applications are, to the best of our knowledge, hardly looked at. Nevertheless, we present here a couple of ideas that are related to our latency reduction mechanisms.

The "Limited Transmit" Mechanism [3] is a TCP mechanism that tries to do more efficient recovery when a connection's congestion window is small, or when a large number of segments are lost in a single transmission window. If the receiver's advertised window allows the transmission of more segments and the amount of outstanding data would remain less than or equal to the congestion window plus two segments, the sender can transmit new data upon the arrival of the first two duplicate ACKs. This happens without changing the congestion window.

In an IETF draft[1], Allman et al. suggest that measures should be taken to recover lost segments when there are too few unacknowledged packets to trigger Fast Retransmit. They propose Early Retransmit (ER), which should reduce waiting times for any of four situations: **1)** the congestion window is still initially small, **2)** it is small because of heavy loss, **3)** flow control limits the send window size, **4)** or the application has no data to send. The draft proposes to act as follows whenever the number of outstanding segments is smaller than 4: if new data is available, it follows Limited Transmit [3], if there is not, it reduces the number of duplicate packets necessary to trigger fast retransmit to as low as 1 depending on the number of unacknowledged segments. Our fast retransmit-triggering mechanism has no stepwise escalation, but is fully applied when there are few packets in flight. This is because we can expect the thin stream to keep its properties throughout its lifetime and also because so many thin-stream applications are interactive with strict latency requirements. Allman et al. try to prevent retransmission timeouts by retransmitting more aggressively, thus keeping the congestion window open. If their limiting conditions change, they still have higher sending rates available. Our main goal is to keep the delivery latency low. We have no motivation to prevent retransmission timeouts in order to keep the congestion window open and retransmit early to reduce application-layer latencies.

The removal of the exponential back-off can of course result in spurious retransmissions when the RTT changes. The proposed method of TCP Santa Cruz [9] uses TCP timestamps and TCP options to determine the copy of a segment that an acknowledgment belongs to and can therefore provide a better RTT estimate. Since the RTT estimate can distinguish multiple packet losses and sudden increases in actual RTT, TCP Santa Cruz can avoid exponential back-off. The ability of Santa Cruz to consider every ACK in RTT estimation has minor effects in our scenario where few packets are generated. The ability to discover the copy of a packet that an ACK refers to would still be desirable but would require receiver-side changes that we avoid.

[1]IETF Draft draft-allman-tcp-early-rexmt-07: Mark Allman, Konstantin Avrachenkov, Urtzi Ayesta, Josh Blanton, Per Hurtig, "Early Retransmit for TCP and SCTP", June 2008, expires December 2008.

V. Test results

We have used well-known applications that generate thin streams in order to test our TCP enhancements. The applications have strict requirements with respect to tolerable latency, and the perceived user satisfaction therefore depends heavily on timely delivery of data.

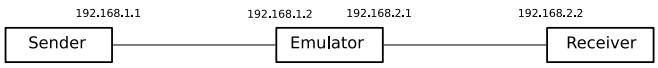


Fig. 2. Network setup for tests.

We used the **netem** [1] emulator with a variety of settings to create loss and delay reflecting a large range of values observed in the Internet. For the presented results, an emulated loss rate of 2 % and an RTT of 130 ms was used as a representative example. These values were chosen to reflect an intercontinental connection between access networks (based on actual measurements and statistics from literature [11]). The network setup is shown in figure 2.

A. Skype

VoIP over the Internet is used by a steadily increasing number of people. We wanted to investigate whether our modifications could improve the perceived quality of such a VoIP session, and chose Skype as the test application. Skype is a popular VoIP program that defaults to UDP for transport, but falls back to TCP if UDP for some reason is blocked. The TCP modifications for thin streams should be able to help reduce latency upon packet loss. Due to the relatively low IAT, the triggering of fast retransmissions by standard mechanisms is possible. The RDB mechanism, however, will be ideal for reducing perceived impact of loss for this data pattern.

When regarding speech in VoIP conferences, differences between each conversation can make it difficult to evaluate one session when compared to another. To have directly comparable data, and to be able to reproduce the results, we chose to use sound clips which we played across the Skype session. We experimented with several different sound clips, both with respect to the numerical results gathered from packet traces, and to the subjective tests that are described in section V-B. A total of three different sound clips were ultimately chosen for this test. Each clip was played two times, one with TCP modifications and one with regular TCP. The sound clip was sent across the Skype connection and the resulting output was recorded at the receiver.

Figure 3 contains the statistical results for one of the sound clips. The cumulative density function (CDF) graphs show the relative difference in latency between the two streams, i.e., how much of the data is received within a certain time. Figure 3(a) shows when the data is delivered to the receiver machine (at the transport layer), and we can see that TCP with modifications delivers lost payload much earlier than standard TCP.

Unfortunately, the transport delivery latency does not say very much about any gains when it comes to the application, and thus perceived quality. TCP is required to provide both reliability and in-order delivery to the application. If a packet loss occurs, the following packets must wait in the receive buffer until the lost segment has been recovered. Thus, there is often a tail of other packets that have been delivered, but must wait before they are all handed to the application.

Figure 3(b) shows the application layer latency, and here we see an even larger gain when using TCP with the modifications. Close to 7% of the data sent with standard TCP had to wait for one or more retransmissions, while only around 2% (the loss rate) of the data sent with the modifications was stuck in the receive buffer. With the average IAT of 34 ms for this stream (see table I), most of the data segments affected by loss were delivered with the next packet. In addition, even though a retransmission occurs, the data is still delivered faster than with standard TCP.

It is also interesting to observe that the application layer latency is very close to the transport layer latency, when the TCP modifications is applied, which means that the implicit delay caused by having to wait for retransmissions is minimal.

B. Skype user test

When taking part in a phone conversation, one of the most important aspects is the sound quality. Distortion and similar artifacts will degrade the user experience, making it more difficult to understand the speaker. We therefore made recordings of Skype conversations played over links with loss and delay, and had a group of people evaluate the perceived quality. The same sound clip was played twice, one time with the TCP modifications and one time with regular TCP.

In order to generate the same original sound for both runs, we played the first minute of the clip. We present results using one speech podcast and two songs. The podcast was chosen because speech is what Skype is designed for, thus the codec should be tuned for it. The songs were chosen because it is easier to notice artifacts when you have a rhythm to relate to. A complicating factor is that Skype encoding sometimes distorts the sound when using TCP, even under perfect network conditions (no loss or delay). All the recordings would, however, be exposed to these irregularities, so the resulting recordings should be directly comparable. Unfortunately, it was not possible to investigate the cause of the distortion further since Skype is a proprietary application.

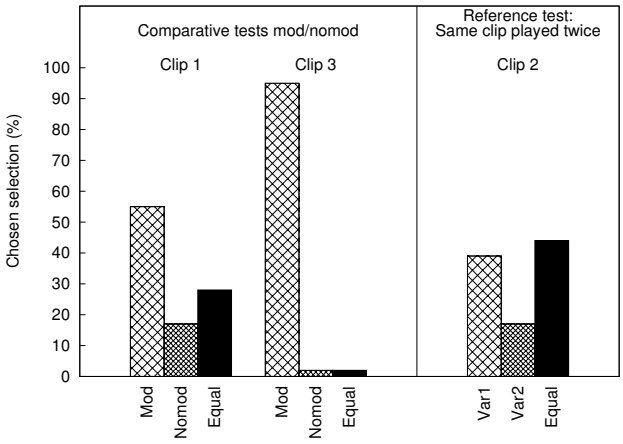
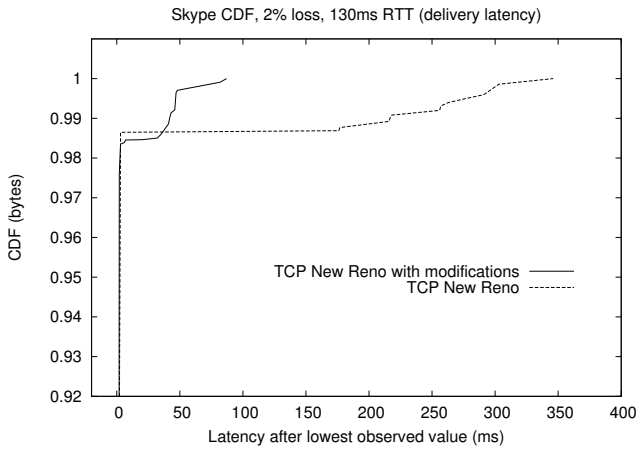


Fig. 4. Preferred sound clips from Skype user tests

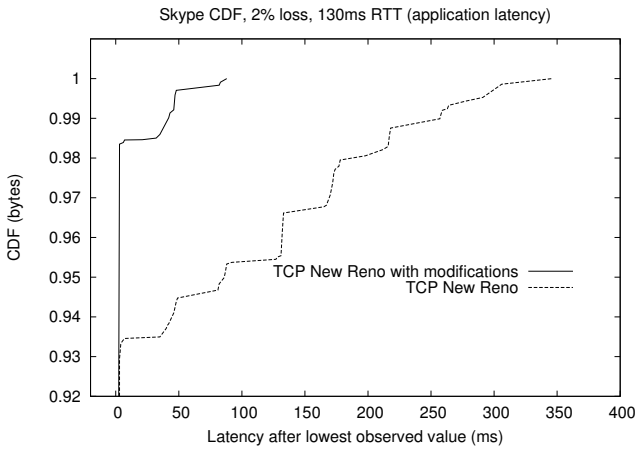
As a reference test, we played the same version of one sound clip twice. This was done to ensure that a "memory effect" does influence the answers overly much (for instance that, after listening to two clips, the listener would prefer the first clip because he/she had "forgotten" the faults observed in that clip.)

All in all we collected 88 votes and the results are shown in figure 4. The recordings made with the modifications were clearly preferred by the users. We were told that the differences in "clip 1" (figure 4), which was the podcast, were small but still noticeable. With clip 3, which was one of the songs, the users commented that the version without the modifications was distinctly suffering in quality compared to the clip run using modified TCP. The test subjects complained about delays, noise, gaps, and others artifacts, and said that it was easy to hear the difference.

In the reference test ("Clip 2" in figure 4), the majority of the test subjects answered that they considered the quality as equal. Of the users that decided on one of the versions of this clip, most of them chose the one that was played first. This may be due to the "memory effect" (discussed above); the listener may have "forgotten" the errors of the first clip. For "Clip 1", the modified version was the second clip to be played. The "memory effect" may here have diminished the results for the modified version of TCP. Even so, a great majority of the test subjects preferred the modified version. For "Clip 3"(figure 4), the order was the opposite (modified TCP first). We can assume that some of the people who chose the



(a) Transport layer latency.



(b) Application layer latency.

Fig. 3. Skype latency

modified TCP version were fooled by the "memory effect". However, the majority of subjects who chose the modified TCP version is so great (95,4%) that we can safely trust the numbers.

C. SSH

The second application that was chosen for user evaluation was secure shell. As described in section II-C, a text-based SSH session represents a typical thin stream. This protocol is widely used as a remote tunnel to administrate servers and remotely edit configuration files. While working in an SSH session, loss can lead to delays that make the editing of text more difficult (e.g. lag between pressing keys and the subsequent screen update). We therefore wanted to analyse the latency for this application and determine whether there is a noticeable difference between using regular TCP and the modified version.

Figure 5 shows the CDFs that resulted from analysing SSH packet traces. A captured SSH session was replayed over the test network (figure 2) for 6 hours. The resulting dumps were analysed with respect to transport and application layer latency. The SSH session statistics (see table I) show that the average IAT is high and the packets are small. This means that all of the modifications are active. For the Skype test, the IAT was relatively low, so RDB was the mechanism that contributed most. With the high IAT for the SSH test, the retransmission mechanisms were in more demand. We can see from figure 5 that the share of data that experience undue delay was also smaller for this stream, when using the modifications. Figure 5(a) shows that 98% of the data was delivered with no extra delay (reflecting the loss rate of 2%). When loss was experienced, the delivery latency for regular TCP severely increases compared to the modified version. When comparing the transport layer latency (figure 5(a)) to the application layer latency (figure 5(b)), we can see that waiting for lost segments is a major delaying factor for as much as 6% of the packets for regular TCP. The high maximum values that we observed was due to the high average IAT for this stream. This resulted in more retransmissions by timeout, and the potential triggering of exponential backoff. For the unmodified TCP test, most of the lost segments were recovered after approximately 330 ms, indicating that retransmission by timeout was the prime mechanism of recovery. For the modified version, a more spread-out pattern reflects the triggering of the modifcations. As for the Skype test, we observe that the delivery latency for the application-layer are almost identical to the latency for the transport layer when applying the modifications.

D. SSH user test

The experience of using a remote text terminal can be severely diminished by network loss. The screen may not be updated with the character that was typed, and it may be difficult to edit the document in a controlled manner. After analysing the SSH latency, we wanted to test if the improvements in application layer latency could be noticed by the user.

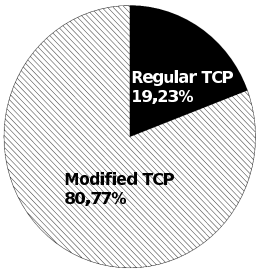
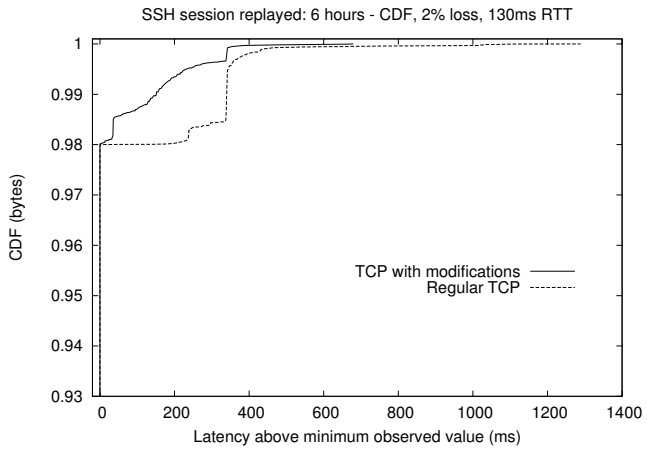
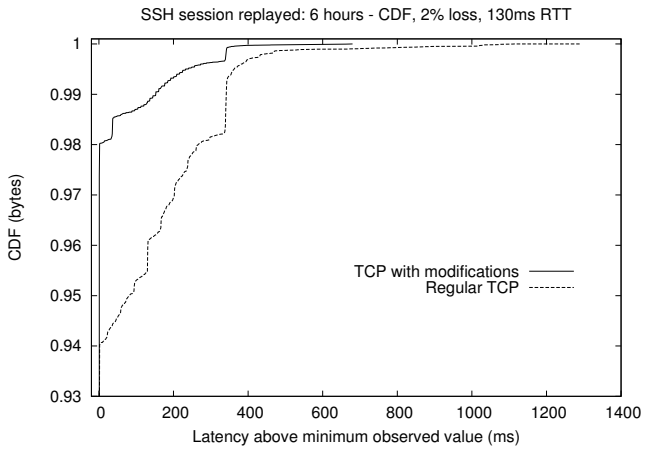


Fig. 6. SSH user test: Preferred connection.

The test network was configured in the same way as for the Skype test. The users opened a command window on the sender computer and initiated a text-based SSH connection to the receiver. Each user then opened a text editor (like "vi" or "emacs") and typed a few sentences. The users were encouraged to try to keep their eyes on the screen while typing in order to observe any irregularities that might occur while typing. In order to make the test applicable to the users that prefer watching the keyboard while writing, a second test was devised. This test consisted of repeatedly hitting the same key, while watching the screen. After the typing, the user was to close the editor and log out of the SSH-session. We then



(a) Transport layer latency.



(b) Application layer latency.

Fig. 5. SSH latency

made a change in the network configuration, and the user repeated the procedure. Afterwards, the test subjects had to decide which of the two sessions they considered to have the best performance. In order to avoid the "memory effect", half of the test subjects took the modified TCP test first, the other half took the regular TCP test first. A total of 26 people participated in this test. All of the participants were familiar with text editing over SSH, and used a text editor that they were comfortable with.

Figure 6 shows how many chose each type of session. The black area represents the percentage of the test subjects who preferred the regular TCP session (19%), while the hachured area represents the ones who preferred the session that used the modified TCP (81%). A frequently occurring comment from the users was that they preferred the connection using the modified TCP because the disturbances that occurred seemed less bursty. That is, if they had written a group of letters, they showed up without a large delay. Another recurring comment was that deleting became more difficult in the sessions using regular TCP because it was easy to delete too many letters at a time.

The large share of users that preferred the session using the modified TCP strongly suggests that the improvement shown in figure 5 can also be experienced at the user level.

The number of participants for the user tests may, statistically, be too small to draw absolute conclusions based on them. Seen in correlation with the measured data, however, we feel that this is a strong indication of the impact of the TCP modifications on the user experience for the tested thin-stream applications.

VI. Discussion

The thin-stream modifications to TCP are based on the concept that the applications in question will never use their fair share of the bandwidth. The price that must be paid for the reduced latency is a greater measure of redundant transmissions. For the exponential backoff- and dupACK modifications, this comes in the form of more aggressive retransmissions. For RDB there is a constant factor of redundancy. The increase in bandwidth, however, is still very small due to the fact that the applications only transmit sporadically (or at a steady low rate).

Tests have shown that the redundancy introduced by our mechanisms is acceptable, at least for removing the exponential backoff and reducing the number of required duplicate acknowledgments. For RDB the redundancy is dependant on the number of bundles that can be made, in other words a connection with a low IAT and high RTT will experience much redundancy. Still, when the loss rate, IAT and RTT display realistic levels, the redundancy introduced with RDB is also acceptable.

Even though we alter the congestion control, tests run so far indicate that fairness is preserved. The amount of spurious transmissions does increase, but the thin (low bandwidth) nature of the stream makes the impact small. The reason for the increased packet rate when RDB is used is that it does not necessarily slow down when loss occurs (since the lost data might arrive in the next packet and be acknowledged before a retransmission is triggered).

If the packets are small enough for bundles to be made, the IAT determines the effect of RDB. If two or more packets can be sent and acknowledged before a timeout, i.e. the IAT is less than the RTO minus the RTT, RDB will reduce both the number of retransmissions and latency. Should the first packet be lost, the data will arrive with the second packet and be acknowledged before a timeout can occur.

If the IAT is larger than the RTO minus the RTT, RDB will still reduce the latency, but will not be able to preempt the retransmission by timeout, thus increasing the total number of transmissions. The retransmitted packets will contain as much unacknowledged data as possible, thus the data will still be delivered faster than with standard TCP.

This paper only discusses changes to TCP. The thin-stream problems will, however, be relevant to most end-to-

end transport protocols that require reliability. The most used approach (for instance when implementing reliability on top of UDP) has been to model retransmission mechanisms after TCP. Viewed in the light of the latency problems of TCP retransmission mechanisms for thin streams, the proposed changes should be considered also for other protocols and applications that implement end-to-end reliability.

VII. Future work

In order to determine the effect of thin-stream modifications on fairness, we have experimented with subjecting regular TCP streams to the competition of modified TCP streams through a bottleneck. The preliminary results show no indication that the modifications inhibit the competing stream whatsoever. It will be more challenging to tell how the streams would perform if a large number of thin-streams using modified TCP should compete for the same bottleneck. In order to know more about these complex fairness scenarios, we are working on simulations of larger networks using thin-stream TCP modifications.

One way to reduce the redundancy of RDB would be to set a limit for the size of bundled packets. This would help avoid the bundling of too many segments on, for instance, a high RTT connection.

The user tests that we present here provide an indication of the experienced gain from using the proposed TCP mechanisms. More extensive tests based on ITU-T standards for subjective user tests can be used to provide further evidence of the effect of the modifications on the experienced performance.

VIII. Conclusion

We have described how applications that generate small packets with high IAT suffer because of the way in which TCP congestion control handles retransmissions. We have evaluated a set of proposed mechanisms with respect to delivery latency, and also gathered user experiences using the same applications with and without the proposed modifications.

The proposed modifications are compliant with TCP standards, and transparent to the receiver. In addition, they are transparent to the application, which makes it possible to use the modifications for existing applications without any changes (as is done in the tests presented).

Our analysis of packet traces shows that the modifications will greatly improve the application layer latency for thin streams when loss occurs. Furthermore, the user surveys indicate that improvements to the users' experience of the tested applications are evident.

References

[1] netem. http://www.linuxfoundation.org/en/Net:Netem, July 2008.

[2] Skype, March 2008. http://www.skype.com.

[3] Allman, M., Balakrishnan, H., and Floyd, S. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042 (Proposed Standard), Jan. 2001.

[4] Allman, M., Paxson, V., and Stevens, W. TCP Congestion Control . RFC 2581 (Proposed Standard), Apr. 1999. Updated by RFC 3390.

[5] Claypool, M., and Claypool, K. Latency and player actions in online games. *Communications of the ACM 49*, 11 (Nov. 2005), 40–45.

[6] Evensen, K., Petlund, A., Griwodz, C., and Halvorsen, P. Redundant bundling in tcp to reduce perceived latency for time-dependent thin streams. *Communications Letters, IEEE 12*, 4 (April 2008), 324–326.

[7] Griwodz, C., and Halvorsen, P. The fun of using TCP for an MMORPG. In *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (May 2006), ACM Press, pp. 1–7.

[8] International Telecommunication Union (ITU-T). One-way Transmission Time, ITU-T Recommendation G.114, 2003.

[9] Parsa, C., and Garcia-Luna-Aceves, J. J. Improving TCP congestion control over internets with heterogeneous transmission media. In *International Conference on Network Protocols (ICNP)* (Nov. 1999), pp. 213–221.

[10] Pedersen, J., Griwodz, C., and Halvorsen, P. Considerations of SCTP retransmission delays for thin streams. In *IEEE Conference on Local Computer Networks (LCN)* (Nov. 2006), pp. 1–12.

[11] Sat, B., and Wah, B. W. Playout scheduling and loss-concealments in voip for optimizing conversational voice communication quality. In *ACM International Multimedia Conference (ACM MM)* (Oct. 2007), pp. 137–146.

[12] Stevens, W. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), Jan. 1997. Obsoleted by RFC 2581.